fluid

Bacco, Giacomo

Citing

Please cite this work if you have used it in your project or in your program.

Giacomo Bacco. *fluid: Free Fluid Flux-Barriers Rotor for Synchronous Reluctance Motor Drawing*. 2018. URL: `https://github.com/gbacco5/fluid`

Contents

Goal

Provide a ready-to-use fully parametric drawing of the Synchronous Reluctance Rotor with fluid flux-barriers.

The scope of this project is the computation of the flux-barriers points. The drawing scripts are for demonstration purposes only.

Requirements

Matlab or Octave or Python (NumPy + SciPy) to compute the points. The points calculation is general, so it could be implemented in any language, but I chose Matlab/Octave because it is my standard interface with FEMM software.

If you do not use FEMM, you can still use the calculation part and make a porting for your CAD engine or FEA software. If you do so, consider contributing to the project adding your interface scripting.

1 Files needed

For Matlab/Octave, the files needed for the computation are
`calc_fluid_barrier.m`,
`GetFSolveOptions.m` and
`isOctave.m`
while for Python it is
`fluid_functions.py`.

Contacts

You can contact me at giacomo.bacco@phd.unipd.it.

If you find a bug, consider opening an issue at `https://github.com/gbacco5/fluid/issues`

Last update: April 22, 2018

2 How to use

Open the file `fluid` and run it.

Change the machine data in the data section. All the variables have a comment next to them.

There are some "hidden" options which should be explained.

1. you can provide personal flux barrier angles, or let the program compute them as the average of the final points C and D at the rotor periphery. This means that you have to always provide the flux-barrier thicknesses and flux-carrier widths. Optionally, you could also provide the electrical flux-barrier angles.

2. by default, the flux-barrier-end is round, so the code solves an additional system to determine the correct locations of the fillet points. You can skip such system declaring

```
1  rotor.barrier_end = 'rect';
```

   and so selecting "rectangular" flux-barrier-end.

3. the inner radial iron ribs are optional, but you are free to provide different widths for every flux-barrier.

```
1  rotor.wrib = [1,2,4]*mm;
```

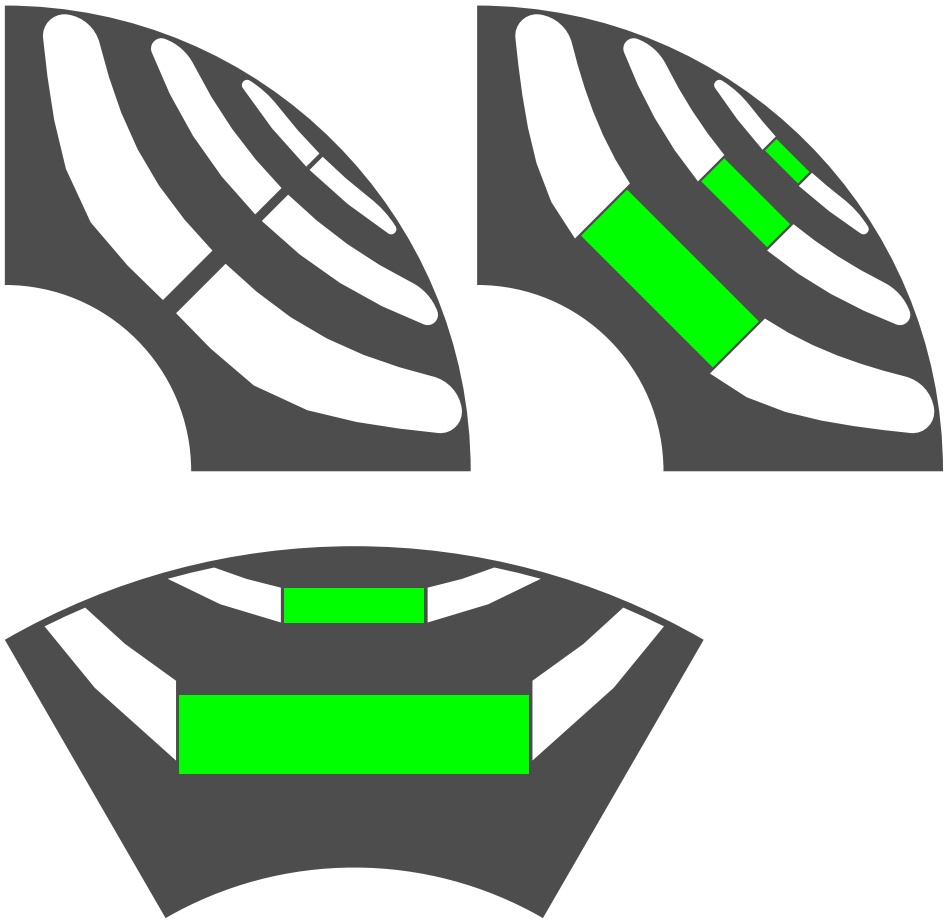
4. if you also input magnet widths, the rib is automatically enlarged to accommodate the magnet, similarly to an IPM (Interior Permanent Magnet) machine.

```
1  rotor.wm = [10,20,40]*mm;
```

   In this case, the output structure `barrier` also contains the location of the magnet base center point.

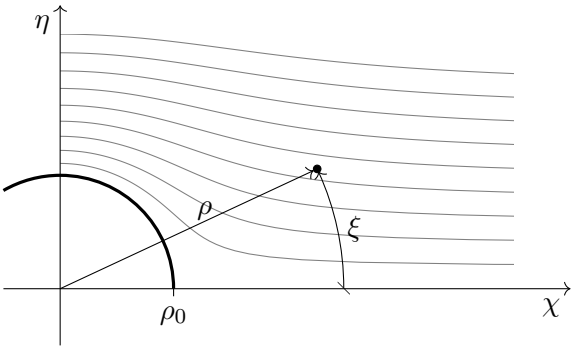
3 Examples

Here are some finished examples based on the output. The drawings are for demonstration purposes only.



4 Theory

4.1 Flow past a cylinder



Let ρ_0 be the radius of the cylinder, ρ, ξ the polar coordinate system in use. One possible solution of this problem have these potential and streamline functions:

$$\phi(\rho, \xi) = \left(\rho + \frac{\rho_0^2}{\rho}\right) \cos \xi \tag{1}$$

$$\psi(\rho, \xi) = \left(\rho - \frac{\rho_0^2}{\rho}\right) \sin \xi \tag{2}$$

Although these equations are deeply coupled, the radius ρ and the phase ξ can be obtained as a function of the other quantities. For our purposes, we use ψ.

$$\rho(\psi, \xi) = \frac{\psi + \sqrt{\psi^2 + 4\rho_0^2 \sin^2 \xi}}{2 \sin \xi} \tag{3}$$

$$\xi(\psi, \rho) = \arcsin\left(\frac{\rho\, \psi}{\rho^2 - \rho_0^2}\right) \tag{4}$$

The velocity field can also be derived through

$$\begin{aligned} v_\rho(\rho, \xi) &= \frac{\partial \phi}{\partial \rho} = \left(1 - \frac{\rho_0^2}{\rho^2}\right) \cos \xi \\ v_\xi(\rho, \xi) &= \frac{1}{\rho} \frac{\partial \phi}{\partial \xi} = -\left(1 + \frac{\rho_0^2}{\rho^2}\right) \sin \xi \end{aligned} \tag{5}$$

4.2 Conformal mapping

From the reference plane, which is equivalent to a two-pole machine, we use a complex map to obtain the quantities in the actual plane. Let p be the number of pole pairs. Then:

$$\begin{aligned} \zeta &\xrightarrow{\mathcal{M}} z = \sqrt[p]{\zeta} \\ \rho\,\mathrm{e}^{\mathrm{j}\xi} &\xrightarrow{\mathcal{M}} r\,\mathrm{e}^{\mathrm{j}\vartheta} = \sqrt[p]{\rho}\,\mathrm{e}^{\mathrm{j}\xi/p} \\ \chi + \mathrm{j}\eta &\xrightarrow{\mathcal{M}} x + \mathrm{j}y \end{aligned} \tag{6}$$

It is easy to find the inverse map:

$$\mathcal{M}\colon \sqrt[p]{\cdot} \qquad \mathcal{M}^{-1}\colon (.)^p \tag{7}$$

In the transformed plane, the velocities have a different expression:

$$\begin{aligned} v_r(r,\vartheta) &= p\left(r^{p-1} - \frac{R_0^{2p}}{r^{p+1}}\right)\cos p\vartheta \\ v_\vartheta(r,\vartheta) &= -p\left(r^{p-1} + \frac{R_0^{2p}}{r^{p+1}}\right)\sin p\vartheta \end{aligned} \tag{8}$$

This vector field is tangent to the streamlines in every point in the transformed plane. In order to work with this field in x, y coordinates, we need a rotational map:

$$\begin{aligned} v_x(r,\vartheta) &= v_r\cos\vartheta - v_\vartheta\sin\vartheta \\ v_\vartheta(r,\vartheta) &= v_r\sin\vartheta + v_\vartheta\cos\vartheta \end{aligned} \tag{9}$$

4.3 Computation of flux-barrier base points

Refer to Figure 1 for the points naming scheme. Keep in mind that A' is not simply the projection of A onto the q-axis, but it represent the original starting point for the barrier sideline, so it lies on the flux-barrier streamline. The same is true for points B', B, C', C, and D', D.

Let the flux-barrier and flux-carrier thicknesses be given. Then the base points for the flux-barriers can be computed easily. Let D_r be the

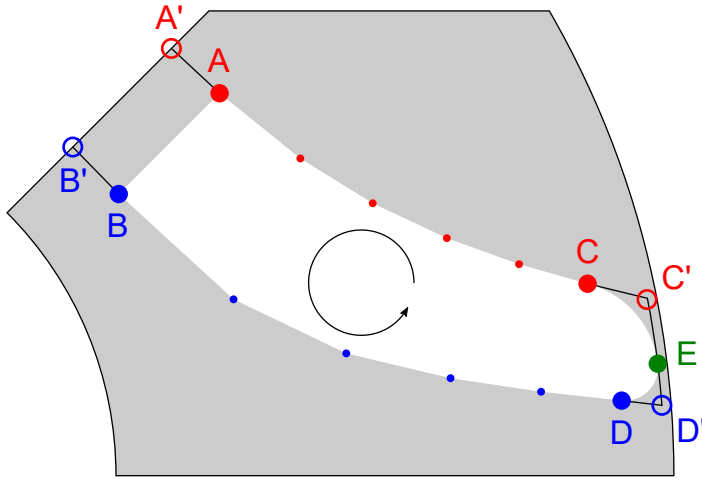


Figure 1: Flux-barrier base points description.

rotor outer diameter, $w_{\mathrm{rib,t}}$ the tangential iron rib width, $w_{\mathrm{c},k}$ the k-th flux-carrier width, and $t_{\mathrm{b},k}$ the k-th flux-barrier thickness.[1] Then

$$\begin{aligned} R_{\mathrm{r}} &= \frac{D_{\mathrm{r}}}{2} - w_{\mathrm{rib,t}} \\ R_{\mathsf{A}'_1} &= R_{\mathrm{r}} - w_{\mathrm{c},1} \\ R_{\mathsf{B}'_1} &= R_{\mathsf{A}'_1} - t_{\mathrm{b},1} \\ &\vdots \end{aligned} \tag{10}$$

where R represents the radius from the origin. So, in general:

$$\begin{aligned} R_{\mathsf{A}'_k} &= R_{\mathsf{B}'_{k-1}} - w_{\mathrm{c},k} \\ R_{\mathsf{B}'_k} &= R_{\mathsf{A}'_{k-1}} - t_{\mathrm{b},k} \end{aligned} \tag{11}$$

with the exception $R_{\mathsf{B}'_0} = R_{\mathrm{r}}$.

[1] You may wonder why the main dimensions of the flux-carrier and flux-barrier differ in the name (width versus thickness). This is due to a choice of mine, because I prefer to refer to width when the flux flows perpendicularly to the dimension, and to thickness when it flows in parallel.

Now we know both the radii and the angle – always $\pi/(2p)$ – of the flux-barrier internal points. So we can compute their respective streamline value.

4.3.1 Magnet insertion

$$w_{\mathrm{rib},k} = w_{\mathrm{rib},k} + w_{\mathrm{m},k}$$

where $w_{\mathrm{m},k}$ is the k-th magnet width.

4.3.2 Central base points

We refer to points A and B. If the rib width is zero $\mathsf{A} \equiv \mathsf{A}'$ and $\mathsf{B} \equiv \mathsf{B}'$.

The line describing the q-axis is

$$\begin{aligned} y &= mx + q \\ m &= \tan\frac{\pi}{2p} \\ q &= \frac{w_{\mathrm{rib}}}{2\cos\frac{\pi}{2p}} \end{aligned} \tag{12}$$

$$\begin{cases} y_{\mathsf{A}} - m x_{\mathsf{A}} - q = 0 \\ x_{\mathsf{A}} - r_{\mathsf{A}}(\psi_{\mathsf{A}'}, \vartheta_{\mathsf{A}})\cos\vartheta_{\mathsf{A}} = 0 \\ y_{\mathsf{A}} - r_{\mathsf{A}}(\psi_{\mathsf{A}'}, \vartheta_{\mathsf{A}})\sin\vartheta_{\mathsf{A}} = 0 \end{cases} \tag{13}$$

where ϑ_{A} is used as the third degree of freedom and r_{A} is then a function of it. The solution of such system can be determined solving the single equation

$$r_{\mathsf{A}}(\psi_{\mathsf{A}'}, \vartheta_{\mathsf{A}})\big(\sin\vartheta_{\mathsf{A}} - m\cos\vartheta_{\mathsf{A}}\big) - q = 0 \tag{14}$$

in the unknown ϑ_{A}. The function $r(\psi, \vartheta)$ is simply

$$r(\psi, \vartheta) = \sqrt[p]{\rho(\psi, \vartheta/p))}$$

The same equation can be written for point B with the proper substitution and repeated for all the flux-barriers.

4.4 Outer base points

We refer to points C, D, and E. If the flux-barrier angle, α_b, is given, then

$$\begin{aligned} x_\mathsf{E} &= R_r \cos(\tfrac{\pi}{2p} - \alpha_\mathrm{b}) \\ y_\mathsf{E} &= R_r \sin(\tfrac{\pi}{2p} - \alpha_\mathrm{b}) \end{aligned} \tag{15}$$

Points C and D results from the connection of the flux-barrier sidelines and point E. This connection should be as smooth as possible in order to avoid dangerous mechanical stress concentrations. We are going to use circular arcs to make this connection. So we impose the tangency between the flux-barrier sideline and the arc, between the arc and the radius through point E. The tangent to the sideline can be obtained through the velocity field described above.

Then we want point C to lay on the flux-barrier sideline. These conditions represent a nonlinear system of 4 equations, in 6 unknowns. So we need two more equations, which are that points C and E belong to the fillet circle with radius R.

$$\begin{cases} x_\mathsf{C} - r_\mathsf{C}(\psi_\mathsf{C}, \vartheta_\mathsf{C}) \cos\vartheta_\mathsf{C} = 0 \\ y_\mathsf{C} - r_\mathsf{C}(\psi_\mathsf{C}, \vartheta_\mathsf{C}) \sin\vartheta_\mathsf{C} = 0 \\ (x_\mathsf{C} - x_{\mathsf{O_C}})^2 + (y_\mathsf{C} - y_{\mathsf{O_C}})^2 - R_\mathsf{EC}^2 = 0 \\ (x_\mathsf{E} - x_{\mathsf{O_C}})^2 + (y_\mathsf{E} - y_{\mathsf{O_C}})^2 - R_\mathsf{EC}^2 = 0 \\ (x_{\mathsf{O_C}} - x_\mathsf{E}) y_\mathsf{E} - (y_{\mathsf{O_C}} - y_\mathsf{E}) x_\mathsf{E} = 0 \\ (x_\mathsf{O} - x_\mathsf{C}) v_x(r_\mathsf{C}, \vartheta_\mathsf{C}) + (y_\mathsf{O} - y_\mathsf{C}) v_y(r_\mathsf{C}, \vartheta_\mathsf{C}) \end{cases} \tag{16}$$

The very same system can be written and solved for point D.

4.5 Flux-barrier sideline points

Consider the top flux-barrier sideline, so the one going from point A to point C. We want to create such sideline using a predetermined number of steps, N_step. From now on, let us call this number N, and N_k for the k-th flux-barrier.

One of the best way to distribute the points along the streamline is to use the potential function, ϕ, defined in Equation 1. We start computing

the potential for points A and C:

$$\phi_{\mathsf{A}} = \phi(\rho_{\mathsf{A}}, \xi_{\mathsf{A}})$$
$$\phi_{\mathsf{C}} = \phi(\rho_{\mathsf{C}}, \xi_{\mathsf{C}})$$

Then, we want to find $N-1$ points along the streamline between points A and C with a uniform distribution of the potential function. We define

$$\Delta\phi_{\mathsf{AC}} = \frac{\phi_{\mathsf{C}} - \phi_{\mathsf{A}}}{N}$$

So we can compute the potentials we are looking for

$$\phi_i = \phi_{\mathsf{A}} + i\Delta\phi_{\mathsf{AC}}\,, \quad i = 1, \ldots, N-1$$

and finally the location of the point with this potential value and the streamline function value required to lie on the flux-barrier sideline. This translates to the following system of equations:

$$\begin{cases} \psi_{\mathsf{AC}} - \psi(\rho, \xi) = 0 \\ \phi_i - \phi(\rho, \xi) = 0 \end{cases} \tag{17}$$

The system is well-defined because there are two unknowns and two independent equations. This system must be solved for every flux-barrier sideline point, for the two sides, and for every flux-barrier.[2]

4.6 Output

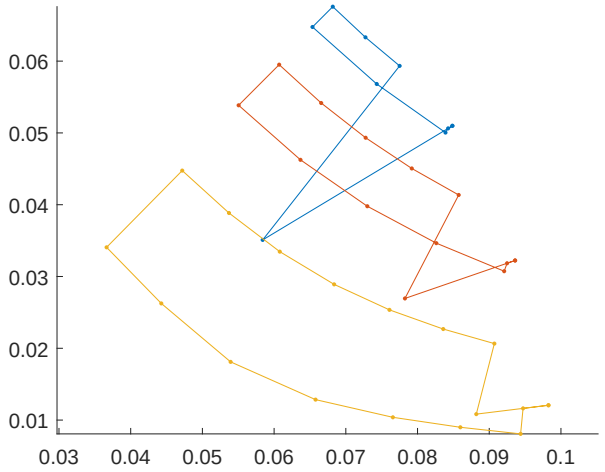
The output of the computation function in Matlab/Octave is one vector of structures (`barrier(:)`) which contains at least two fields (`X` and `Y`). The X vector is made in this way:

$$X = \begin{bmatrix} x_{\mathsf{E}} & x_{\mathsf{O_C}} & x_{\mathsf{C}} & x_{\mathsf{AC}_{N_{\text{step}}-1}} & \cdots & x_{\mathsf{AC}_1} & x_{\mathsf{A}} & \searrow \\ x_{\mathsf{B}} & x_{\mathsf{BD}_1} & \cdots & x_{\mathsf{BD}_{N_{\text{step}}-1}} & x_{\mathsf{D}} & x_{\mathsf{O_D}} & x_{\mathsf{E}} & \end{bmatrix}^{\mathrm{T}}$$

and similarly the Y vector. So the points are ordered starting from the point E and then moving counter-clockwise until E is reached again.

[2] In Matlab/Octave, the "for every flux-barrier sideline point" loop has been vectorized, while the two sides has been manually split.

4.7 Example of Matlab/Octave plot



Here is an example of a Matlab/Octave output plot. The V-shaped lines represent the radii of the fillet arcs, which were not worth to be shown in Matlab/Octave.

5 Code

5.1 Main file

```
% FLUID
% Free Fluid Flux-Barriers Rotor for Synchronous Reluctance Motor Drawing
%
% Bacco, Giacomo 2018

clear all; close all; clc;
addpath('draw','tools');

%% DATA
rotor.p = 2; % number of pole pairs
mm = 1e-3; % millimeters
rotor.De = 200*mm; % [m], rotor outer diameter

rotor.Nb = 3; % number of flux-barriers
rotor.tb = [4 8 15]*mm; % flux-barrier thicknesses
rotor.wc = [3 7 12 10]*mm; % flux-carrier widths
rotor.Nstep = 3*[2, 4, 6]; % number of steps to draw the flux-barrier side
rotor.wrib_t = 1*mm; % [m], tangential iron rib width

% you can input flux-barrier angles or let the program compute them
% rotor.barrier_angles_el = [14,26,38]*2; % [deg], electrical flux-barrier angles
% rotor.barrier_end = 'rect'; % choose 'rect' or comment

% you can define the rib width or comment
rotor.wrib = [0,1,1]*mm; % [m], radial iron rib widths
% You can define the magnet width or comment
% rotor.wm = [10,20,40]*mm;

%% barrier points computation
barrier = calc_fluid_barrier(rotor);

%% simple matlab plot
figure
```

```
hold all
axis equal
for bkk = 1:rotor.Nb
  plot(barrier(bkk).X, barrier(bkk).Y, '.-')
end
if isfield(rotor,'wm')
  RM = [barrier(:).Rm];
  thM = pi/2/rotor.p;
  Xm = RM.*cos(thM);
  Ym = RM.*sin(thM);
  plot(Xm, Ym, 'ko')
end
axis auto

%% FEMM drawing
try
  openfemm(1)
  newdocument(0);

  draw_fluid_barrier(barrier);

catch
  disp('FEMM not available.');

end
```

5.2 Calc fluid barrier

```
function barrier = calc_fluid_barrier(r)
% CALC_FLUID_BARRIER computes the flux-barrier points
    along the streamline
% function.

%% DATA
global deb

Dr = r.De; % [m], rotor outer diameter
ScalingFactor = 1/( 10^(round(log10(Dr))) );
% ScalingFactor = 1;
Dr = Dr*ScalingFactor;

p = r.p; % number of pole pairs
Nb = r.Nb; % number of flux-barriers
tb = r.tb*ScalingFactor; % flux-barrier widths
wc = r.wc*ScalingFactor; % flux-carrier widths
Nstep = r.Nstep; % number of steps to draw the flux-
    barrier side

wrib_t = r.wrib_t*ScalingFactor; % [m], tangential iron rib
    width

if isfield(r,'barrier_angles_el')
  barrier_angles_el = r.barrier_angles_el; % [deg], electrical
     flux-barrier angles
  AutoBarrierEndCalc = 0;
else
  barrier_angles_el = zeros(1,Nb);
  AutoBarrierEndCalc = 1;
end

if isfield(r,'wm')
  wm = r.wm*ScalingFactor;
else
  wm = 0;
end
if isfield(r,'wrib')
  wrib = r.wrib*ScalingFactor + wm; % [m], radial iron rib
```

```
    widths
else
  wrib = zeros(1,Nb) + wm;
end

Dend = Dr - 2*wrib_t; % [m], flux-barrier end diameter
Dsh = Dend - 2*( sum(tb) + sum(wc) ); % [m], shaft diameter
R0 = Dsh/2; % [m], shaft radius
barrier_angles = barrier_angles_el/p; % [deg], flux-barrier
    angles
if isfield(r,'barrier_end')
  barrier_end = r.barrier_end;
else
  barrier_end = '';
end

%% IMPLICIT FUNCTIONS
% definition of fluid past a cylinder functions
psi_fluid = @(rho,xi,rho0) (rho.^2 - rho0^2)./rho.*sin(xi);
phi_fluid = @(rho,xi,rho0) (rho.^2 + rho0^2)./rho.*cos(xi);
xi_fluid  = @(psi,rho,rho0) asin(psi.*rho./(rho.^2 - rho0^2));
rho_fluid = @(psi,xi,rho0) ( psi + sqrt(psi.^2 + 4*sin(xi).^2*
    rho0^2) )./(2*sin(xi));

r_map = @(rho) rho.^(1/p);
th_map = @(xi) xi./p;
rho_map = @(r) r.^p;
xi_map = @(th) th.*p;

vr = @(r,th,R0) p*(r.^(p-1) - R0^(2*p)./r.^(p+1)).*cos(p*th);
vt = @(r,th,R0) -p*(r.^(p-1) + R0^(2*p)./r.^(p+1)).*sin(p*th);
vx = @(vr_v,vth_v,th) vr_v.*cos(th) - vth_v.*sin(th);
vy = @(vr_v,vth_v,th) vr_v.*sin(th) + vth_v.*cos(th);

%% Precomputations
rho0 = rho_map(R0);

%% Central base points
RAprime = Dend(1)/2 - [0, cumsum( tb(1:end-1))] - cumsum(wc(1:
    end-1)); % top
RBprime = RAprime - tb; % bottom
```

```
73  te_qAxis = pi/(2*p); % q-axis angle in rotor reference
        frame

75  % get A' and B' considering rib and magnet widths
76  mCentral = tan(te_qAxis); % slope
77  qCentral = repmat( -wrib/2/cos(te_qAxis), 1, 2); % intercept

79  psiCentralPtA = psi_fluid(rho_map(RAprime), xi_map(te_qAxis),
        rho0);
80  psiCentralPtB = psi_fluid(rho_map(RBprime), xi_map(te_qAxis),
        rho0);
81  psiCentralPt = [psiCentralPtA, psiCentralPtB];
82  psiA = psiCentralPtA;
83  psiB = psiCentralPtB;

85  CentralPt_Eq = @(th) ...
86    r_map( rho_fluid(psiCentralPt, xi_map(th), rho0) ).*...
87    ( sin(th) - mCentral*cos(th) ) - qCentral;

89  if deb == 1
90    options.Display = 'iter'; % turn off folve display
91  else
92    options.Display = 'off'; % turn off folve display
93  end
94  options.Algorithm = 'levenberg-marquardt'; % non-square
        systems
95  options.FunctionTolerance = 10*eps;
96  options.StepTolerance = 1e4*eps;

98  X0 = repmat(te_qAxis,1,2*Nb);
99  options = GetFSolveOptions(options);
100 teAB = fsolve(CentralPt_Eq, X0, options);
101 teA = teAB(1:Nb);
102 teB = teAB(Nb+1:end);
103 RA = r_map( rho_fluid(psiA, xi_map(teA), rho0) );
104 RB = r_map( rho_fluid(psiB, xi_map(teB), rho0) );

106 % magnet central base point radius computation
107 RAsecond = RA.*cos(te_qAxis - teA);
108 RBsecond = RB.*cos(te_qAxis - teB);
```

```
110  Rmag = (RAprime + RAsecond + RBprime + RBsecond)/4;

112  %% Outer base points C,D preparation
113  RCprime = Dend/2;
114  teCprime = th_map( xi_fluid(psiA, rho_map(RCprime), rho0) );
115  xCprime = Dend/2.*cos(teCprime);
116  yCprime = Dend/2.*sin(teCprime);

118  RDprime = Dend/2;
119  teDprime = th_map( xi_fluid(psiB, rho_map(RDprime), rho0) );
120  xDprime = Dend/2.*cos(teDprime);
121  yDprime = Dend/2.*sin(teDprime);

123  if AutoBarrierEndCalc
124    teE = (teCprime + teDprime)/2;
125    aphE = pi/2/p - teE;
126    barrier_angles = 180/pi*aphE;
127    barrier_angles_el = p*barrier_angles;
128  else
129    aphE = barrier_angles*pi/180;
130    teE = pi/2/p - aphE;
131  end
132  xE = Dend/2.*cos(teE);
133  yE = Dend/2.*sin(teE);

135  %% Outer base points C (top)
136  if strcmp(barrier_end, 'rect')
137    RC = RCprime;
138    teC = teCprime;
139    xC = xCprime;
140    yC = yCprime;
141    xOC = xC;
142    yOC = yC;

144  else
145    options.Algorithm = 'trust-region-dogleg'; % non-square
         systems

147    BarrierEndSystem = @(th,xd,yd,xo,yo,R) ...
148      [xd - r_map(rho_fluid(psiA', p*th, rho0)).*cos( th )
149      yd - r_map(rho_fluid(psiA', p*th, rho0)).*sin( th )
```

```
150       (xd - xo).^2 + (yd - yo).^2 - R.^2
151       (xE' - xo).^2 + (yE' - yo).^2 - R.^2
152       (xo - xd).*vx( vr( r_map(rho_fluid(psiA', p*th, rho0)),th,R0
           ), vt( r_map(rho_fluid(psiA', p*th, rho0)) ,th,R0 ), th) +
          (yo - yd).*vy(  vr( r_map(rho_fluid(psiA', p*th, rho0)),th,
          R0 ), vt( r_map(rho_fluid(psiA', p*th, rho0)) ,th,R0 ), th)
153       (xo - xE').*yE' - (yo - yE').*xE'
154       %   th - xi_fluid((rho_fluid(p*th, psiA', rho0)),
          psiA', rho0)/p % serve?
155       ];

157     X0 = [ 1.5*teE', 0.9*xE', 0.9*yE', 0.8*xE', 0.8*yE', 0.25*xE
          '];
158     % X0 = [ aph_b, 0, 0, 0, 0, 0];
159     X = fsolve( @(x) BarrierEndSystem( x(:,1),x(:,2),x(:,3),x(:,4)
          ,x(:,5),x(:,6) ), X0, options);

161     xOC = X(:,4)';
162     yOC = X(:,5)';
163     xC = X(:,2)';
164     yC = X(:,3)';
165     RC = hypot(xC, yC);
166     teC = atan2(yC, xC);
167   end

169   %% Outer base points D (bottom)
170   if strcmp(barrier_end, 'rect')
171     RD = RDprime;
172     teD = teDprime;
173     xD = xDprime;
174     yD = yDprime;
175     xOD = xD;
176     yOD = yD;

178   else
179     options.Algorithm = 'levenberg-marquardt'; % non-square
          systems

181     BarrierEndSystem = @(th,xd,yd,xo,yo,R) ...
182       [xd - r_map(rho_fluid(psiB', p*th, rho0)).*cos( th )
183       yd - r_map(rho_fluid(psiB', p*th, rho0)).*sin( th )
```

```
184       (xd - xo).^2 + (yd - yo).^2 - R.^2
185       (xE' - xo).^2 + (yE' - yo).^2 - R.^2
186       (xo - xd).*vx( vr( r_map(rho_fluid(psiB', p*th, rho0)),th,R0
           ), vt( r_map(rho_fluid(psiB', p*th, rho0)) ,th,R0 ), th) +
          (yo - yd).*vy(  vr( r_map(rho_fluid(psiB', p*th, rho0)),th,
          R0 ), vt( r_map(rho_fluid(psiB', p*th, rho0)) ,th,R0 ), th)
187       (xo - xE').*yE' - (yo - yE').*xE'
188       %   th - xi_fluid((rho_fluid(p*th, psi_d, rho0)),
          psi_d, rho0)/p % serve?
189       ];

191     X0 = [ 0.8*teE', 0.8*xE', 0.8*yE', xE'*.9, yE'*.9, xE'*.2];
192     X = fsolve( @(x) BarrierEndSystem( x(:,1),x(:,2),x(:,3),x(:,4)
          ,x(:,5),x(:,6) ), X0, options);

194     xOD = X(:,4)';
195     yOD = X(:,5)';
196     xD = X(:,2)';
197     yD = X(:,3)';
198     RD = hypot(xD, yD);
199     teD = atan2(yD, xD);
200   end

202   %% Flux-barrier points
203   % We already have the potentials of the two flux-barrier
          sidelines
204   phiA = phi_fluid( rho_map(RA), xi_map(teA), rho0);
205   phiB = phi_fluid( rho_map(RB), xi_map(teB), rho0);

207   phiC = phi_fluid( rho_map(RC), xi_map(teC), rho0);
208   phiD = phi_fluid( rho_map(RD), xi_map(teD), rho0);

210   %% Code for single Nstep
211   % dphiAC = (phiC - phiAprime)./Nstep;
212   % dphiBD = (phiD - phiBprime)./Nstep;
213   %
214   % % we create the matrix of potentials phi needed for
          points intersections
215   % PhiAC = phiAprime + cumsum( repmat(dphiAC, Nstep - 1,
          1) );
216   % PhiBD = phiBprime + cumsum( repmat(dphiBD, Nstep - 1,
```

```
    1) );
%
% PhiAC_vec = reshape(PhiAC, numel(PhiAC), 1);
% PhiBD_vec = reshape(PhiBD, numel(PhiBD), 1);
% PsiAC_vec = reshape( repmat( psiA, Nstep-1, 1), numel(
    PhiAC), 1 );
% PsiBD_vec = reshape( repmat( psiB, Nstep-1, 1), numel(
    PhiBD), 1 );
%
% % we find all the barrier points along the streamline
% PsiPhi = @(rho,xi, psi,phi, rho0) ...
%   [psi - psi_fluid(rho, xi, rho0)
%    phi - phi_fluid(rho, xi, rho0)];
%
% X0 = [repmat(rho0*1.1, numel(PhiAC_vec), 1), repmat(pi
    /4, numel(PhiAC_vec), 1)];
% RhoXi_AC = fsolve( @(x) PsiPhi( x(:,1),x(:,2),
    PsiAC_vec, PhiAC_vec, rho0 ), X0, options);
% RhoXi_BD = fsolve( @(x) PsiPhi( x(:,1),x(:,2),
    PsiBD_vec, PhiBD_vec, rho0 ), X0, options);
%
% R_AC = reshape( r_map(RhoXi_AC(:,1)), Nstep-1, Nb );
% te_AC = reshape( th_map(RhoXi_AC(:,2)), Nstep-1, Nb );
% R_BD = reshape( r_map(RhoXi_BD(:,1)), Nstep-1, Nb );
% te_BD = reshape( th_map(RhoXi_BD(:,2)), Nstep-1, Nb );

%% Code for different Nsteps
% we find all the barrier points along the streamline
PsiPhi = @(rho,xi, psi,phi, rho0) ...
  [psi - psi_fluid(rho, xi, rho0)
  phi - phi_fluid(rho, xi, rho0)];

% barrier(Nb).R_AC = 0;
% barrier(Nb).R_BD = 0;
% barrier(Nb).te_AC = 0;
% barrier(Nb).te_BD = 0;
barrier(Nb) = struct;

for bkk = 1:Nb
  dphiAC = (phiC(bkk) - phiA(bkk))./Nstep(bkk);
  dphiBD = (phiD(bkk) - phiB(bkk))./Nstep(bkk);
```

```
252     % we create the matrix of potentials phi needed for
          points intersections
253     PhiAC = phiA(bkk) + cumsum( repmat(dphiAC', Nstep(bkk) - 1, 1)
          );
254     PhiBD = phiB(bkk) + cumsum( repmat(dphiBD', Nstep(bkk) - 1, 1)
          );
255     PsiAC = repmat( psiA(bkk), Nstep(bkk)-1, 1);
256     PsiBD = repmat( psiB(bkk), Nstep(bkk)-1, 1);

258   % 1st try
259   %   X0 = [repmat(rho0*1.1, numel(PhiAC), 1), repmat(pi
        /4, numel(PhiAC), 1)];
260   % 2nd try
261   %   X0 = [repmat(rho0*1.1, numel(PhiAC), 1), repmat(
        xi_map(teE(bkk)), numel(PhiAC), 1)];
262   % 3rd try
263     X0 = [linspace(rho0, Dend/2, numel(PhiAC))', linspace(pi/4,
          xi_map(teE(bkk)), numel(PhiAC))'];
264     RhoXi_AC = fsolve( @(x) PsiPhi( x(:,1),x(:,2), PsiAC, PhiAC,
          rho0 ), X0, options);
265     RhoXi_BD = fsolve( @(x) PsiPhi( x(:,1),x(:,2), PsiBD, PhiBD,
          rho0 ), X0, options);

267     R_AC = r_map(RhoXi_AC(:,1));
268     te_AC = th_map(RhoXi_AC(:,2));
269     R_BD = r_map(RhoXi_BD(:,1));
270     te_BD = th_map(RhoXi_BD(:,2));

272     if deb
273       barrier(bkk).R_AC = R_AC/ScalingFactor;
274       barrier(bkk).R_BD = R_BD/ScalingFactor;
275       barrier(bkk).te_AC = te_AC;
276       barrier(bkk).te_BD = te_BD;
277     end

279     % output of points
280   %   barrier(bkk).Zeta = [...
281     Zeta = [...
282       % top side
283       xE(bkk) + 1j*yE(bkk)
284       xOC(bkk) + 1j*yOC(bkk)
```

```
285       xC(bkk) + 1j*yC(bkk)
286       flipud( R_AC.*exp(1j*te_AC) )
287       RA(bkk).*exp(1j*teA(bkk))
288       % bottom side
289       RB(bkk).*exp(1j*teB(bkk))
290       R_BD.*exp(1j*te_BD)
291       xD(bkk) + 1j*yD(bkk)
292       xOD(bkk) + 1j*yOD(bkk)
293       xE(bkk) + 1j*yE(bkk)
294       ]/ScalingFactor;

296     barrier(bkk).X = real(Zeta);
297     barrier(bkk).Y = imag(Zeta);

299     % magnet central base point
300     barrier(bkk).Rm = Rmag(bkk)/ScalingFactor;

302   end

304   %% plot
305   if deb

307     % draw the rotor
308     figure
309     hold on
310     tt = linspace(0,pi/p,50);
311     plot(R0/ScalingFactor*cos(tt), R0/ScalingFactor*sin(tt), 'k');
312     plot(Dr/2/ScalingFactor*cos(tt), Dr/2/ScalingFactor*sin(tt), '
          k');
313     axis equal
314     % plot the flux-barrier central point
315     plot(RA/ScalingFactor.*exp(1j*teA), 'rd')
316     plot(RB/ScalingFactor.*exp(1j*teB), 'bo')

318     plot(xE/ScalingFactor, yE/ScalingFactor,'ko')

320     plot(xOC/ScalingFactor, yOC/ScalingFactor,'go')
321     plot(xC/ScalingFactor, yC/ScalingFactor,'ro')
322     plot(xOD/ScalingFactor, yOD/ScalingFactor,'co')
323     plot(xD/ScalingFactor, yD/ScalingFactor,'bo')
```

```
325     %
326     % plot(R_AC.*exp(j*te_AC),'r.-')
327     % plot(R_BD.*exp(j*te_BD),'b.-')

329     for bkk = 1:Nb
330       % plot flux-barrier sideline points
331       plot(barrier(bkk).R_AC.*exp(1j*barrier(bkk).te_AC),'r.-')
332       plot(barrier(bkk).R_BD.*exp(1j*barrier(bkk).te_BD),'b.-')

334       % plot all the complete flux-barrier
335       plot(barrier(bkk).X, barrier(bkk).Y, '.-')
336     end
337     pause(1e-3)

339   end

341   end
```

5.3 Draw fluid barrier

```
function draw_fluid_barrier(b)

for bkk = 1:length(b)
  xE = b(bkk).X(1);
  yE = b(bkk).Y(1);
  xEOC = b(bkk).X(2);
  yEOC = b(bkk).Y(2);
  xC = b(bkk).X(3);
  yC = b(bkk).Y(3);

  xD = b(bkk).X(end-2);
  yD = b(bkk).Y(end-2);
  xDOE = b(bkk).X(end-1);
  yDOE = b(bkk).Y(end-1);

  X = b(bkk).X(3:end-2);
  Y = b(bkk).Y(3:end-2);

  mi_drawpolyline([X, Y])

  if xEOC == xC && yEOC == yC
    mi_drawline(xE,yE, xC,yC)
  else
    mi_draw_arc(xE,yE, xEOC,yEOC, xC,yC, 1)
  end

  if xDOE == xD && yDOE == yD
    mi_drawline(xD,yD, xE,yE)
  else
    mi_draw_arc(xD,yD, xDOE,yDOE, xE,yE, 1)
  end

end

end
```